ESc 101: Fundamentals of Computing

Lecture 14

Feb 1, 2010

Outline

Data Structure for Negative Numbers

- A number is stored, as usual, in an array of `char`, with one digit per element.
- The order is least significant digit first.
- For storing the sign, the possibilities are:
  - Use one element of the array to record the sign.
  - Use a separate variable to store the sign.

Data Structure for Negative Numbers

- A number is stored, as usual, in an array of `char`, with one digit per element.
- The order is least significant digit first.
- For storing the sign, the possibilities are:
  - Use one element of the array to record the sign.
  - Use a separate variable to store the sign.

Data Structure for Negative Numbers

- A number is stored, as usual, in an array of `char`, with one digit per element.
- The order is least significant digit first.
- For storing the sign, the possibilities are:
  - Use one element of the array to record the sign.
  - Use a separate variable to store the sign.

Data Structure for Negative Numbers

- A number is stored, as usual, in an array of `char`, with one digit per element.
- The order is least significant digit first.
- For storing the sign, the possibilities are:
  - Use one element of the array to record the sign.
  - Use a separate variable to store the sign.

Storing Sign in Array

- An obvious way is to store it as the last element of the array.
- Adding two numbers will now require checking the signs of numbers and then performing either addition or subtraction.
- If one number is negative and other positive, we also need to check which is larger and subtract accordingly.

Storing Sign in Array

- An obvious way is to store it as the last element of the array.
- Adding two numbers will now require checking the signs of numbers and then performing either addition or subtraction.
- If one number is negative and other positive, we also need to check which is larger and subtract accordingly.

Storing Sign in Array

- An obvious way is to store it as the last element of the array.
- Adding two numbers will now require checking the signs of numbers and then performing either addition or subtraction.
- If one number is negative and other positive, we also need to check which is larger and subtract accordingly.

Storing Sign Separately

- The same process needs to be followed for adding two numbers.

To keep it simple, let us store the sign in the array.

Storing Sign Separately

- The same process needs to be followed for adding two numbers.

To keep it simple, let us store the sign in the array.

Algorithm for Adding Numbers

```
Algorithm add_numbers(n, m)
{
1. If both are positive, then add directly.
2. If both are negative, remove the signs, add,
    and then give the result negative sign.
3. Otherwise, remove the signs, and subtract the
    smaller number from the larger one, and give
    the result the sign of larger number.
}
```

A Better Data Structure

- A number can have at most `SIZE` digits.
- In other words, a number is between $-10^{\texttt{SIZE}} + 1$ to $+10^{\texttt{SIZE}} - 1$.
- Consider these numbers modulo $2 * 10^{\texttt{SIZE}}$:
  - Numbers between 0 and $10^{\texttt{SIZE}} - 1$ remain the same.
  - Number -1 becomes $2 * 10^{\texttt{SIZE}} - 1$.
  - Number $-10^{\texttt{SIZE}} + 1$ becomes $2 * 10^{\texttt{SIZE}} - 10^{\texttt{SIZE}} + 1 = 10^{\texttt{SIZE}} + 1$.
  - All other negative numbers are in between $10^{\texttt{SIZE}} + 1$ and $2 * 10^{\texttt{SIZE}} - 1$.
- All numbers are now positive!
- Number $10^{\texttt{SIZE}}$ corresponds to $-10^{\texttt{SIZE}}$ but will never be input.

A Better Data Structure

- A number can have at most `SIZE` digits.
- In other words, a number is between $-10^{\texttt{SIZE}} + 1$ to $+10^{\texttt{SIZE}} - 1$.
- Consider these numbers modulo $2 * 10^{\texttt{SIZE}}$:
  - Numbers between 0 and $10^{\texttt{SIZE}} - 1$ remain the same.
  - Number -1 becomes $2 * 10^{\texttt{SIZE}} - 1$.
  - Number $-10^{\texttt{SIZE}} + 1$ becomes $2 * 10^{\texttt{SIZE}} - 10^{\texttt{SIZE}} + 1 = 10^{\texttt{SIZE}} + 1$.
  - All other negative numbers are in between $10^{\texttt{SIZE}} + 1$ and $2 * 10^{\texttt{SIZE}} - 1$.
- All numbers are now positive!
- Number $10^{\texttt{SIZE}}$ corresponds to $-10^{\texttt{SIZE}}$ but will never be input.

A Better Data Structure

- A number can have at most SIZE digits.
- In other words, a number is between $-10^{\text{SIZE}} + 1$ to $+10^{\text{SIZE}} - 1$.
- Consider these numbers modulo $2 * 10^{\text{SIZE}}$:
  - Numbers between 0 and $10^{\text{SIZE}} - 1$ remain the same.
  - Number -1 becomes $2 * 10^{\text{SIZE}} - 1$.
  - Number $-10^{\text{SIZE}} + 1$ becomes $2 * 10^{\text{SIZE}} - 10^{\text{SIZE}} + 1 = 10^{\text{SIZE}} + 1$.
  - All other negative numbers are in between $10^{\text{SIZE}} + 1$ and $2 * 10^{\text{SIZE}} - 1$.
- All numbers are now positive!
- Number 10^{SIZE} corresponds to -10^{SIZE} but will never be input.

A Better Data Structure

- A number can have at most SIZE digits.
- In other words, a number is between $-10^{\text{SIZE}} + 1$ to $+10^{\text{SIZE}} - 1$.
- Consider these numbers modulo $2 * 10^{\text{SIZE}}$:
  - Numbers between 0 and $10^{\text{SIZE}} - 1$ remain the same.
  - Number -1 becomes $2 * 10^{\text{SIZE}} - 1$.
  - Number $-10^{\text{SIZE}} + 1$ becomes $2 * 10^{\text{SIZE}} - 10^{\text{SIZE}} + 1 = 10^{\text{SIZE}} + 1$.
  - All other negative numbers are in between $10^{\text{SIZE}} + 1$ and $2 * 10^{\text{SIZE}} - 1$.
- All numbers are now positive!
- Number 10^{SIZE} corresponds to -10^{SIZE} but will never be input.

A Better Data Structure

- A number can have at most SIZE digits.
- In other words, a number is between $-10^{\text{SIZE}} + 1$ to $+10^{\text{SIZE}} - 1$.
- Consider these numbers modulo $2 * 10^{\text{SIZE}}$:
  - Numbers between 0 and $10^{\text{SIZE}} - 1$ remain the same.
  - Number -1 becomes $2 * 10^{\text{SIZE}} - 1$.
  - Number $-10^{\text{SIZE}} + 1$ becomes $2 * 10^{\text{SIZE}} - 10^{\text{SIZE}} + 1 = 10^{\text{SIZE}} + 1$.
  - All other negative numbers are in between $10^{\text{SIZE}} + 1$ and $2 * 10^{\text{SIZE}} - 1$.
- All numbers are now positive!
- Number 10^{SIZE} corresponds to -10^{SIZE} but will never be input.

A Better Data Structure

- A number can have at most SIZE digits.
- In other words, a number is between $-10^{\text{SIZE}} + 1$ to $+10^{\text{SIZE}} - 1$.
- Consider these numbers modulo $2 * 10^{\text{SIZE}}$:
  - Numbers between 0 and $10^{\text{SIZE}} - 1$ remain the same.
  - Number -1 becomes $2 * 10^{\text{SIZE}} - 1$.
  - Number $-10^{\text{SIZE}} + 1$ becomes $2 * 10^{\text{SIZE}} - 10^{\text{SIZE}} + 1 = 10^{\text{SIZE}} + 1$.
  - All other negative numbers are in between $10^{\text{SIZE}} + 1$ and $2 * 10^{\text{SIZE}} - 1$.
- All numbers are now positive!
- Number 10^{SIZE} corresponds to -10^{SIZE} but will never be input.

A Better Data Structure

- A number can have at most SIZE digits.
- In other words, a number is between $-10^{\text{SIZE}} + 1$ to $+10^{\text{SIZE}} - 1$.
- Consider these numbers modulo $2 * 10^{\text{SIZE}}$:
  - Numbers between 0 and $10^{\text{SIZE}} - 1$ remain the same.
  - Number -1 becomes $2 * 10^{\text{SIZE}} - 1$.
  - Number $-10^{\text{SIZE}} + 1$ becomes $2 * 10^{\text{SIZE}} - 10^{\text{SIZE}} + 1 = 10^{\text{SIZE}} + 1$.
  - All other negative numbers are in between $10^{\text{SIZE}} + 1$ and $2 * 10^{\text{SIZE}} - 1$.
- All numbers are now positive!
- Number 10^{SIZE} corresponds to -10^{SIZE} but will never be input.

Adding Numbers

- Numbers are added modulo $2 * 10^{\texttt{SIZE}}$.
- If the result is between 0 and $10^{\texttt{SIZE}} - 1$, the result is positive number.
- If the number is between $10^{\texttt{SIZE}}$ and $2 * 10^{\texttt{SIZE}} - 1$, the result is negative number.
- We need to add only positive number now!
- The addition needs to be done modulo $2 * 10^{\texttt{SIZE}}$.
- This is simpler than handling different cases as earlier.

Adding Numbers

- Numbers are added modulo $2 * 10^{\texttt{SIZE}}$.
- If the result is between 0 and $10^{\texttt{SIZE}} - 1$, the result is positive number.
- If the number is between $10^{\texttt{SIZE}}$ and $2 * 10^{\texttt{SIZE}} - 1$, the result is negative number.
- We need to add only positive number now!
- The addition needs to be done modulo $2 * 10^{\texttt{SIZE}}$.
- This is simpler than handling different cases as earlier.

Adding Numbers

- Numbers are added modulo $2 * 10^{\texttt{SIZE}}$.
- If the result is between 0 and $10^{\texttt{SIZE}} - 1$, the result is positive number.
- If the number is between $10^{\texttt{SIZE}}$ and $2 * 10^{\texttt{SIZE}} - 1$, the result is negative number.
- We need to add only positive number now!
- The addition needs to be done modulo $2 * 10^{\texttt{SIZE}}$.
- This is simpler than handling different cases as earlier.

Adding Numbers

- Numbers are added modulo $2 * 10^{\texttt{SIZE}}$.
- If the result is between 0 and $10^{\texttt{SIZE}} - 1$, the result is positive number.
- If the number is between $10^{\texttt{SIZE}}$ and $2 * 10^{\texttt{SIZE}} - 1$, the result is negative number.
- We need to add only positive number now!
- The addition needs to be done modulo $2 * 10^{\texttt{SIZE}}$.
- This is simpler than handling different cases as earlier.

Adding Numbers

- Numbers are added modulo $2 * 10^{\texttt{SIZE}}$.
- If the result is between 0 and $10^{\texttt{SIZE}} - 1$, the result is positive number.
- If the number is between $10^{\texttt{SIZE}}$ and $2 * 10^{\texttt{SIZE}} - 1$, the result is negative number.
- We need to add only positive number now!
- The addition needs to be done modulo $2 * 10^{\texttt{SIZE}}$.
- This is simpler than handling different cases as earlier.

Adding Numbers

- Numbers are added modulo $2 * 10^{\texttt{SIZE}}$.
- If the result is between 0 and $10^{\texttt{SIZE}} - 1$, the result is positive number.
- If the number is between $10^{\texttt{SIZE}}$ and $2 * 10^{\texttt{SIZE}} - 1$, the result is negative number.
- We need to add only positive number now!
- The addition needs to be done modulo $2 * 10^{\texttt{SIZE}}$.
- This is simpler than handling different cases as earlier.

read_number()

```
/* Reads a number with up to SIZE many digits.
 * Stores the number modulo 2*10^SIZE with least significant
 * digit first.
 */
int read_number(char number[])
{
    char symbol; /* Stores current input symbol */
    char temp[SIZE]; /* temporary storage for numbers */
    int size; /* stores the number of digits in input */
```

read_number()

```
printf("Input a number of at most %d digits: ", SIZE);
symbol = getchar(); /* read first symbol */
if (symbol == '-') { /* negative number */
   number[SIZE] = 1;
   symbol = getchar(); /* read the first digit */
}
else /* positive number */
  number[SIZE] = 0;
```

read_number()

```
for (size = 0; 1; size++) {
   if ((symbol < '0') || (symbol > '9')) /* not a digit */
      break;
   if (size == SIZE) { /* input too large */
      printf("Input too large: number should be at most %d
      return 0;
   }
   temp[size] = symbol - '0';
   symbol = getchar(); /* read next symbol */
}
```

read_number()

```
    /* Store number in reverse order,
    * leaving the sign in place
  */
    int i;

    for (i = 0; i < size; i++)
        number[i] = temp[size-1-i];
    for (i = size; i < SIZE; i++)
        number[i] = 0;
    /* Convert to modular representation */
    number2modular(number);
    return 1;
}
```

number2modular()

```
/* Converts the given number to a number modulo 2*10^SIZE. */
void number2modular(char number[])
{
   int i;

   if (number[SIZE] == 0) /* positive number */
      return;
  /* Subtract from 2*10^SIZE */
   for (i = 0; number[i] == 0; i++);  /* skip zeros */

   if (i == SIZE) { /* number is -0 */
      number[SIZE] = 0; /* Remove the -ve sign */
      return;
   }
```

void: no return value

number2modular()

```
    /* Non-zero digit. Subtract from 10 */
    number[i] = 10 - number[i];

    /* Subtract remaining digits from 9 */
    for (i++; i < SIZE; i++)
        number[i] = 9 - number[i];

    return;
}
```

add_numbers()

```
void add_numbers(char num1[], char num2[], char num3[])
{
   int carry; /* Stores the carry value */

   for (i = 0, carry = 0; i <= SIZE; i++) {
      num3[i] = num1[i] + num2[i] + carry;
      if (num3[i] > 9) { /* new carry created */
         num3[i] = num3[i] - 10;
         carry = 1;
      }
      else /* no carry created */
         carry = 0;
   }
```

add_numbers()

```
    if (num3[SIZE] == 2) { /*  sum too large */
      num3[SIZE] = 0; /* go modulo 2*10^SIZE */
    }
    return;
}
```

output_number()

```
/* Outputs the given number. It first converts the number
 * from its representation modulo 2*10^SIZE to normal.
 */
int output_number(char number[])
{
    int i;
    /* Convert to normal representation */
    modular2number(number);
    /* Skip the leading zeroes */
    for (i = SIZE-1; i >= 0; i--)
        if (number[i] > 0)
            break;
```

output_number()

```
    if (i == 0) { /* the sum is zero! */
        printf("The sum is: 0\n");
        return;
    }
    /* Non-zero number */
    printf("The sum is: ");
    if (number[SIZE] == 1) /* negative number */
        printf(''-''); /* output - sign */

    for (; i >= 0; i--)
        putchar(number[i]+'0');
    printf("\n");

    return;
}
```

main()

```
/* Numbers are stored modulo 2*10^SIZE.
 * Negative numbers are in the range 10^SIZE+1 to 2*10^SIZE-1.
 * Positive numbers are in the range 0 to 10^SIZE-1.
 * All the operations are done modulo 2*10^SIZE.
 */
main()
{
    char number1[SIZE+1]; /* stores first number */
    char number2[SIZE+1]; /* stores second number */
    char number3[SIZE+1]; /* stores the result */
```

main()

```
    /* Read first number */
    if (read_number(number1) == 0) /* error */
        return;
    /* Read second number */
    if (read_number(number2) == 0) /* error */
        return;
    /* Add the two numbers */
    add_numbers(number1,number2,number3);
    /* Output the result */
    output_number(number3);
}
```